Linguagem C#

Programação Visual

Sumário

- Conceitos básicos da linguagem C# em comparação com as linguagens Java e C++.
- **Conceitos avançados da linguagem C#**
- A linguagem integrada de interrogação de dados – LINQ

C# - Conceitos Avançados

Sessão 1 – Inicialização de objetos

Sintaxe de inicialização de objetos, variáveis de tipo implicito e variáveis de tipos anónimos.

C# - Sintaxe de inicialização de objetos

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
    public string Color { get; set; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Point() { }
    public void Display()
    {
        Console.WriteLine("({0},{1})", X, Y);
    }
}
```

C# - Sintaxe de inicialização de objetos

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
    public string Color { get; set; }

    public Point() { }

    public Point(int x, int y)
    {  X = x;  Y = y;  }

    public void Display()
    {  Console.WriteLine("({0},{1})", X, Y); }
}
```

```
Point pt1 = new Point();
pt1.X = 10;
pt1.Y = 10;
pt1.Color = "blue";
pt1.Display();

Point pt2 = new Point(20, 20);
pt2.Color = "blue";
pt2.Display();
```

```
Point pt1 = new Point { X = 10, Y = 10, Color = "blue" };
pt1.Display();

Point pt2 = new Point(20, 20) { Color = "blue" };
pt2.Display();
```

C# - Sintaxe de inicialização de objetos

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
    public string Color { get; set; }

    public Point() { }

    public Point(int x, int y)
    {  X = x;  Y = y;  }

    public void Display()
    {  Console.WriteLine("({0},{1})", X, Y); }
}
```

```
List<Point> pts = new List<Point>()
{
    new Point{X=10, Y=10, Color="blue"},
    new Point(20, 20){Color="blue"},
    null
};
```

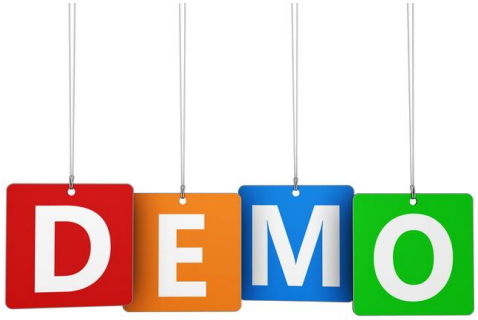
C# - Variáveis de tipo implicito

```
double val = 25.0;

List<string> nomes = new List<string>();
```

```
var val = 25.0;

var nomes = new List<string>();
```



C# - Conceitos Avançados

Sessão 2 – strings interpoladas e métodos de extensão

Strings interpoladas e métodos de extensão.

C# - Strings interpoladas

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
    public string Color { get; set; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Point() { }
    public void Display()
    {
        Console.WriteLine("({0},{1})", X, Y);
    }
}
```

Semelhante ao **`string.Format()`** do Java

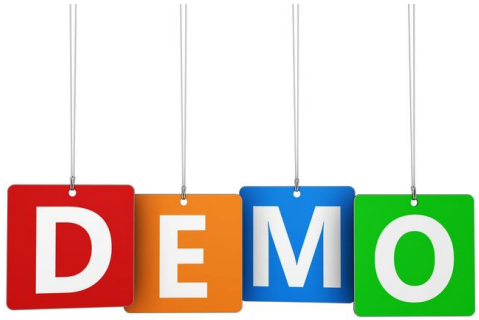
C# - Strings interpoladas

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
    public string Color { get; set; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Point() { }
    public void Display()
    {
        Console.WriteLine( $"({X},{Y})");
    }
}
```

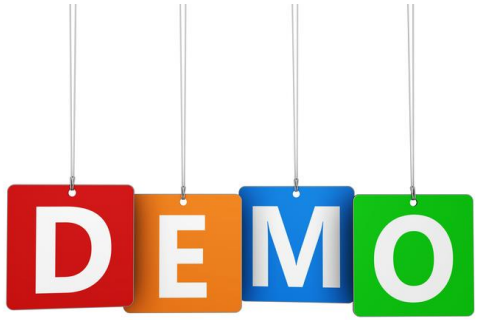
String interpolada



C# - Métodos de extensão

```
public static class MyExtensionMethods
{
    public static bool IsNumeric(this string s)
    {
        float output;
        return float.TryParse(s, out output);
    }
}
```

```
string test = "4.0";
if (test.IsNumeric())
    Console.WriteLine("Yes");
else
    Console.WriteLine("No");
```



C# - Conceitos Avançados

Sessão 3 – Lidar com o valor null

Tipos *nullable*, operadores *null coalescing* e operadores condicionais *null*.

C# - Tipos *nullable*

```
int? i = null;

if (i.HasValue)
    Console.WriteLine(i.Value); //ou (i)
else
    Console.WriteLine("Null");
```

```
Nullable<int> i = null;

if (i.HasValue)
    Console.WriteLine(i.Value); //ou (i)
else
    Console.WriteLine("Null");
```

C# - Operadores *null-coalescing*

Utilização dos operadores

```
int? a = null;

int b = a ?? -1;

Console.WriteLine(b);  // output: -1
```

```
List<int> numbers = null;

numbers ??= new List<int>();
```

Código equivalente

```
int? a = null;

int b;
if (a == null)
    b = -1;
else
    b = a.Value;

Console.WriteLine(b);  // output: -1
```

```
List<int> numbers = null;

if (numbers == null)
    numbers = new List<int>();
```

C# - Operador condicional *null*

Utilização dos operadores

```
A?.B?.Do(C);
```

```
Tipo X = A?.B?[C];
```

Código equivalente

```
if (A != null)
    if (B != null)
        A.B.Do(C);
```

```
Tipo X = null;
if (A != null)
    if (B[C] != null)
        Tipo X = B[C];
```

C# - Conceitos Avançados

Sessão 4 – Expressões e sintaxe Lambda

Expressões Lambda, métodos e propriedades *expression-bodied*.

C# – Expressões Lambda

Método

```
public int Add(int a, int b)
{
    return a + b;
}
```

Expressão Lambda

```
(int a, int b) =>   {
                        return a + b;
                    }
```

C# – Expressões Lambda

Método

```
public int Add(int a, int b)
{
    return a + b;
}
```

Expressão Lambda

```
(a, b) =>   {
                return a + b;
            }
```

C# – Expressões Lambda

Método

```
public int Add(int a, int b)
{
    return a + b;
}
```

Expressão Lambda

```
(a, b) => a + b
```

C# – Expressões Lambda

Método

```
public int Add(int a, int b)
{
    return a + b;
}
```

Expressão Lambda

```
(a, b) =>  a + b
```

```
public void DisplayA (int a)
{
    Console.WriteLine("a= " + a);
}
```

```
(a) =>  Console.WriteLine("a= " + a)
```

C# – Expressões Lambda

Método

Expressão Lambda

```
public int Add(int a, int b)
{
    return a + b;
}
```

```
(a, b) =>  a + b
```

```
public void DisplayA (int a)
{
    Console.WriteLine("a= " + a);
}
```

```
a =>  Console.WriteLine("a= " + a)
```

```
public void Hello()
{
    Console.WriteLine("Hello");
}
```

```
() =>  Console.WriteLine("Hello")
```

C# – Métodos *Expression Bodied*

Método

Expression - Bodied

```
public int Add(int a, int b)
{
    return a + b;
}
```

⟺

```
public int Add(int a, int b) => a + b;
```

```
public void DisplayA (int a)
{
    Console.WriteLine("a= " + a);
}
```

⟺

```
public void DisplayA (int a) => Console.WriteLine("a= " + a);
```

```
public void Hello()
{
    Console.WriteLine("Hello");
}
```

⟺

```
public void Hello() => Console.WriteLine("Hello");
```

C# – Propriedades *Expression Bodied*

Propriedade

```
public class Circle
{
    private int radius;

    public int Radius
    {
        get { return radius; }
        set { radius = value; }
    }

    public double Area
    {
        get { return Math.PI*Radius*Radius; }
    }
}
```

Expression - Bodied

```
public class Circle
{
    private int radius;

    public int Radius
    {
        get => radius;
        set => radius = value;
    }

    public double Area
    {
        get { return Math.PI*Radius*Radius; }
    }
}
```

C# – Propriedades *Expression Bodied*

Propriedade

```
public class Circle
{
    private int radius;

    public int Radius
    {
        get { return radius; }
        set { radius = value; }
    }

    public double Area
    {
        get { return Math.PI*Radius*Radius; }
    }
}
```

Expression - Bodied

```
public class Circle
{
    private int radius;

    public int Radius
    {
        get => radius;
        set => radius = value;
    }

    public double Area
    {
        get => Math.PI*Radius*Radius;
    }
}
```

C# – Propriedades *Expression Bodied*

Propriedade

```
public class Circle
{
    private int radius;

    public int Radius
    {
        get { return radius; }
        set { radius = value; }
    }

    public double Area
    {
        get { return Math.PI*Radius*Radius; }
    }
}
```

Expression - Bodied

```
public class Circle
{
    private int radius;

    public int Radius
    {
        get => radius;
        set => radius = value;
    }

    public double Area => Math.PI*Radius*Radius;

}
```

C# - Conceitos Avançados

Sessão 5 – Delegates e eventos

Delegates e eventos.

C# – Delegates

Delegate

- É um novo tipo cujas variáveis podem guardar métodos
  - Os métodos guardados num dado delegate têm todos a mesma assinatura e tipo de retorno

```
public delegate      Notify
```

C# – Delegates

Delegate

- É um novo tipo cujas variáveis podem guardar métodos
  - Os métodos guardados num dado delegate têm todos a mesma assinatura e tipo de retorno

```
public delegate void Notify(decimal newPrice);
```

C# – Delegates

Delegate

- É um novo tipo cujas variáveis podem guardar métodos
  - Os métodos guardados num dado delegate têm todos a mesma assinatura e tipo de retorno

```
public delegate void Notify(decimal newPrice);
```

```
public class Stock
{
 // ...
     public void PriceChanged(decimal price)
     {
         //...
     }
 }
```

```
Stock stock = new Stock();

Notify notify = stock.PriceChanged;

notify(123m); // corre o método guardado
```

C# – Delegates

Delegate

- É um novo tipo cujas variáveis podem guardar métodos
  - Os métodos guardados num dado delegate têm todos a mesma assinatura e tipo de retorno
  - Podem-se acrescentar métodos usando o operador **+=** ou remover métodos usando o operador **-=**

```
public delegate void Notify(decimal newPrice);
```

```
Stock stock = new Stock();

Notify notify = stock.PriceChanged;

notify += (a) => Console.WriteLine("Preço alerado");

notify(123m); // corre os métodos guardado
```

C# – Eventos

Event

- Serve para criar implicitamente um delegate dentro de uma classe.
  - Os eventos são públicos
  - Podem-se acrescentar métodos usando o operador **+=** ou remover métodos usando o operador **-=**

```
public delegate void Notify(decimal newPrice);
```

```
public class Item
{
    public Notify priceChanged;

    private decimal price;

    public decimal Price
    {
        get => price;
        set
        {
            if (value != price)
                priceChanged(value);
            price = value;
        }
    }
}
```

```
Item item = new Item { Price = 25m };

item.priceChanged += a => Console.WriteLine("Preço Mudou!");

item.Price = 23m;
```

Problema:
Atributo público

C# – Eventos

Event

- Serve para criar implicitamente um delegate dentro de uma classe.
  - Os eventos são públicos
  - Podem-se acrescentar métodos usando o operador **+=** ou remover métodos usando o operador **-=**

```
public delegate void Notify(decimal newPrice);
```

```
public class Item
{
    public event Notify PriceChanged;

    private decimal price;

    public decimal Price
    {
        get => price;
        set
        {
            if (value != price)
                PriceChanged(value);
            price = value;
        }
    }
}
```

```
Item item = new Item { Price = 25m };

item.PriceChanged += a => Console.WriteLine("Preço Mudou!");

item.Price = 23m;
```

C# – Eventos

Event

- Serve para criar implicitamente um delegate dentro de uma classe.
  - Os eventos são públicos
  - Podem-se acrescentar métodos usando o operador **+=** ou remover métodos usando o operador **-=**

```
public delegate void Notify(decimal newPrice);
```

```
public class Item
{
    private Notify priceChanged;

    public event Notify PriceChanged
    {
        add
        {
            priceChanged += value;
        }
        remove
        {
            priceChanged -= value;
        }
    }
}
```

```
Item item = new Item { Price = 25m };

item.PriceChanged += a => Console.WriteLine("Preço Mudou!");

item.Price = 23m;
```

C# – Eventos

Delegate e **Event**